

Software Architecture and Design

© 2009 Anthony J. Lattanze
Institute for Software Research
Carnegie Mellon University



Architectural Structures II

Nguyen Duc Man, Msc
DTU International School
Email: mannd@duytan.edu.vn
Cell: 0904 235 945



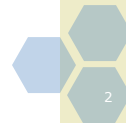
1



Lecture Topics



- ❖ **Introduce the concept of architectural structures as it relates to software intensive systems**
- ❖ **Provide examples of how software structures can help architect design software intensive systems**







2



Perspective, Structures, Relationships, and Views



More on this later when we discuss documentation

 Perspective	 Example Structures	 Example Relationships	 Views
Dynamic	Processes Treads :	Dataflow Events :	Component and Connector
Static	Layers Code Modules :	Depends Uses :	Module
Physical	Computers Sensors :	Serial Line Wireless :	Allocation

© 2000 Jeffrey J. Leffman

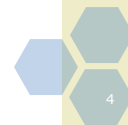
3



Back To The Definition...



- ❖ “The software architecture of a program or computing system is the structure or structures of the system, **which comprise the software elements, the externally visible properties of those elements,** and the relationships among them.”



4



Recall our Analysis Framework



- ❖ **To analyze structures use the following as a framework as a guide. For any given structure, analyze it in terms of:**

- Perspective
- topological arrangement
- elements and relationships
- level of abstraction
- Semantics
- typical usage
- qualities promoted and inhibited

- ❖ **Lets look at each in detail...**



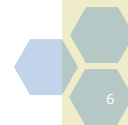
Perspective



- ❖ **The type of structures that are “apparent” depend upon perspective**

- static perspective: classes, modules, libraries, uses, depends,...
- dynamic perspective: processes, threads, data flow, events,...
- physical perspective: computers, sensors, networks, routers,...

- ❖ **Perspective is important in reasoning about and understanding the properties of a structure**





Topological Arrangement



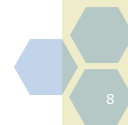
- ❖ **The topological arrangement is often the canonical picture that is often the first indication of presence of a particular structure**
- ❖ **The topology depicts the elements, relationships, and their visible pattern of arrangement**
- ❖ **Again, what elements and relationships that are depicted depend upon perspective**



Elements and Relationships



- ❖ **Elements are typically the focus of functionality, whose type is bound by perspective:**
 - process, thread, object, library,...
- ❖ **Between elements are relationships that indicate that the element interacts in some way, dependent upon perspective:**
 - uses, dataflow, inherits, network, events,...

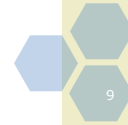




Level of Abstraction



- ❖ **Architecture is design, but not all design is architectural**
 - Typically for architectural design structures, we focus on:
 - the roles and responsibilities of elements
 - rules for their interconnection
 - the qualities promoted/inhibited
 - Architectural structures tend to:
 - be language neutral
 - avoid structured/object oriented paradigms
 - avoid specific application technologies



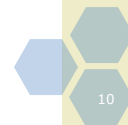
9



Semantics



- ❖ **Semantics are the rules for using a structure**
- ❖ **If the semantics are adhered to, the structure will deliver certain predictable properties**
- ❖ **If the semantics violated, the properties that the structure can deliver, may not be**



10



Typical Usage



- ❖ **What kinds of situations, domains, or problem is this particular pattern used to address?**
- ❖ **A mental catalogue of problems and structures that address them is a use base of knowledge from which to design software intensive systems**



11



Qualities Promoted/Inhibited



- ❖ **Each structure that we select/design will promote some quality attribute properties and inhibit others**
- ❖ **How a system responds to quality attribute stimuli is largely dependent upon how the system is structured**
- ❖ **As architects, we must constantly and explicitly evaluate the various tradeoff inherent in the design choices we make**
- ❖ **Our goal is to achieve the optimal balance that best meets the needs of the stakeholders**



12



Example: Layers – 1



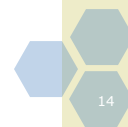
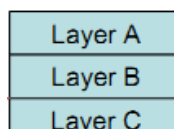
- ❖ **The layers architectural pattern is used to structure applications that can be decomposed into groups of elements in each of which address a different level of abstraction**
- ❖ **Elements**
 - elements are layers and are typically libraries or collections of related services
 - specific and related functional responsibilities are assigned to each layer
- ❖ **Relations**
 - relationships between layers is often implicit and is usually call-return oriented



Example: Layers – 2



- ❖ **Perspective and view**
 - code oriented; usually module view, where each layer is a separate code library. At runtime (once compiled) the layers disappear
- ❖ **Topology**
 - layers are positions adjacent to one another
 - relationships between layers are not usually explicitly depicted – this is OK, if the pattern semantics are adhered to





Example: Layers – 3



❖ **Semantics**

- Layers can only call on services of the next lower layer; that is, each layer provides services to the layer above it and calls on services provided by the layer below it

❖ **Example properties promoted**

- modifiability, portability, reusability

❖ **Example properties inhibited**

- performance, size of implementation; that is executable code built using this pattern will be larger



15

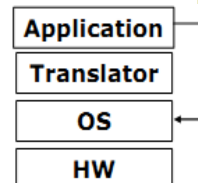


Semantics: Why Bother?



Let's consider a specific example...

Lets assume that portability is an important business goal, so I select a decoupling mechanism like layers...



This structure has semantics, properties it promotes and inhibits. Knowing these enables early analysis of the properties.

Assume that we chose this structure *without any analysis* and we find out that the system is too slow,... what can we do?

We can bridge layers,... but what happens to portability?

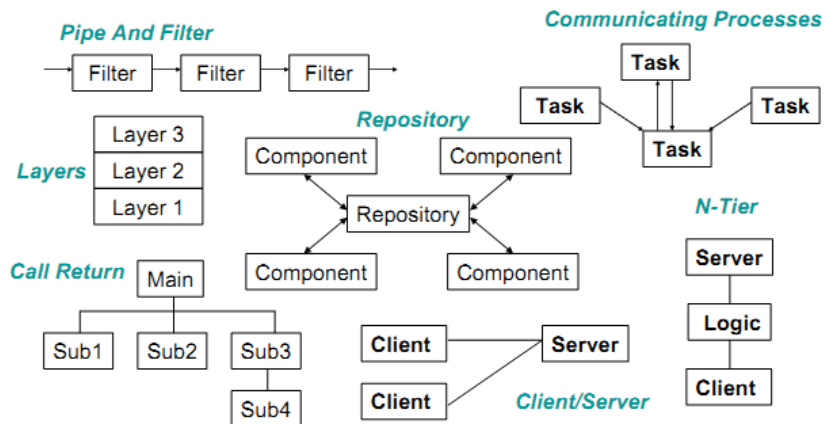
These decisions are only good or bad depending upon the relative importance of portability and performance – *but without the proper expertise, early analysis is impossible!*



16



A Few Basic Patterns



17



A Final Word on Structures



- ❖ We will examine many of these structures in more detail, but it would be impossible to study all of them in the time we have
- ❖ There are numerous books that discuss architectural and design patterns
- ❖ The most important skill to have is how to analyze the structure of a pattern to understand how it can be utilized in practice



18



More Information?



- Various texts can provide pointers to pattern oriented structures. In particular (see references slide):
 - Shaw and Garlan (styles)
 - Buschmann (patterns)
 - Lattanze (analysis framework)
 - Bass, Clements, and Kazman (tactics)
 - Gamma et al. (object oriented patterns)
- The language and terminology will not be consistent, *hence the importance of a framework like this for analyzing structures*



19



The Real World



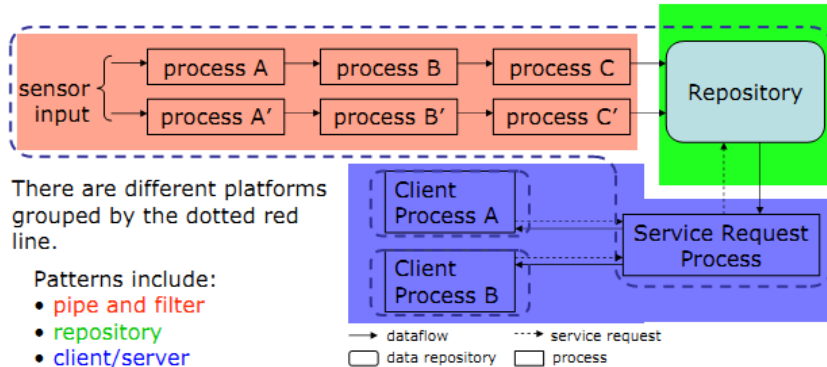
- In the real world, systems have many patterns that interact or are related to one another in a few ways:
 - Separate and interacting patterns
 - cases where patterns are visibly distinct and interact in a clear unambiguous way
 - perspectives must match
 - Hierarchical encapsulated patterns
 - cases where patterns contain other patterns
 - perspectives must match
 - Simultaneous patterns
 - cases where one pattern/patterns from one perspective transforms into others
 - perspectives must match



20



Separate and Interacting



The patterns are separate and distinct, with clear points of interface and interaction.



21



Comments



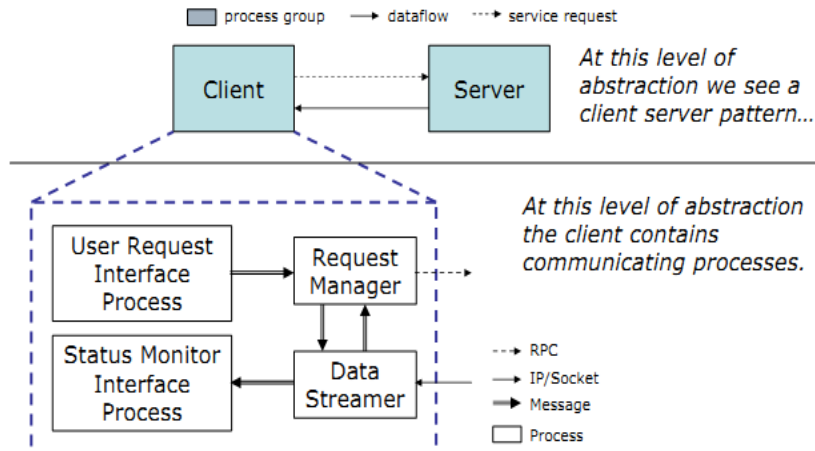
- ❖ **This is not a complete architecture design, it is an initial decomposition – much design remains**
 - patterns can help guide the initial decomposition
 - Note that where the patterns interface and interact should be places of interest, and possibly concern, for the architect
- ❖ **Note that the perspective is consistent**
 - pipe and filter, client/server, repository are represented from the dynamic perspective
 - there is an allocation of structures to platforms here, but the view is not cluttered with hardware details



22



Hierarchical and Encapsulated



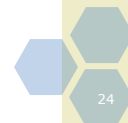
23



Comments – 1



- ❖ **Note that all of the elements here are of the same dynamic perspective**
- ❖ **There are two patterns shown here**
 - client-server
 - communicating processes residing inside the client
- ❖ **There are cases (in the next example) where a system exhibits multiple patterns from different perspectives – this is not the case here**
- ❖ **This example is a matter of more patterns being exposed with more details**



24



Comments – 2



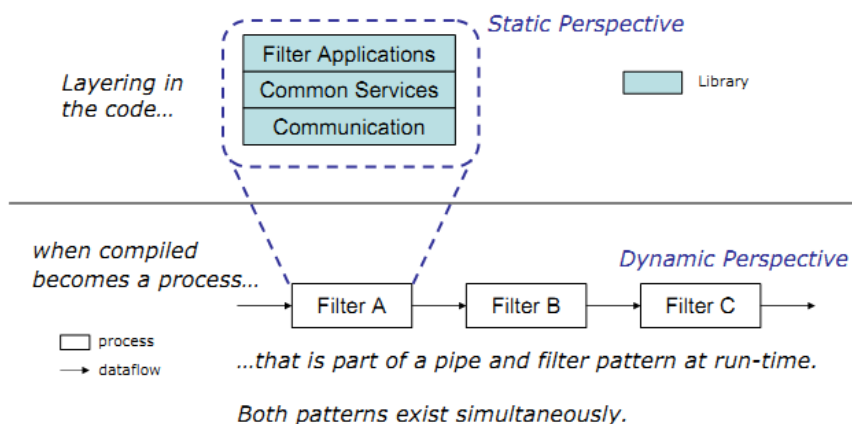
- ❖ **This example also illustrates that architecture design can and often is hierarchical – that is they often have multiple levels of detail**
 - At each level of architectural design, elements are further decomposed and responsibilities are assigned as externally visible properties to the elements
 - When we can no longer partition the system and only concern ourselves with defining external element properties, the architecture design may be done – then we look inward and transition to detailed design



25



Simultaneous



26



Comments



- ❖ In real systems it is common for patterns to be connected as ensembles in all of these ways:
 - Separate and Interacting, Hierarchical and Encapsulated, and Simultaneous
- ❖ If architects can tease apart the constituent structures of an ensemble and reason about them and their interconnection in isolation, it can help with overall systemic analysis
 - It's possible to identify potential problem areas, opportunities, and leads to a deeper, more thorough analysis of the design



27



Domain Specific Patterns – 1



- ❖ **Often it makes sense (economically) to codify an architecture pattern that typifies a family of systems/products reoccurring problems unique in our own organizations and domains**
 - promotes commonality among products
 - facilitates reuse and product lines
 - reduces learning curve and time to market
- ❖ **Examples from industry include ERP/CRM, avionics, vehicle data and control systems, telecommunications, and many more**



28



Domain Specific Patterns – 2



- ❖ **Repeating structures can lead to reference architectures which are knowledge models**
- ❖ **Reference architectures can be used to create implementation frameworks**
- ❖ **Implementation frameworks are not the product architecture but provide a significant economic advantage by reducing...**
 - the amount of time spent designing from scratch, coding and testing
 - the number of defects introduced into the product



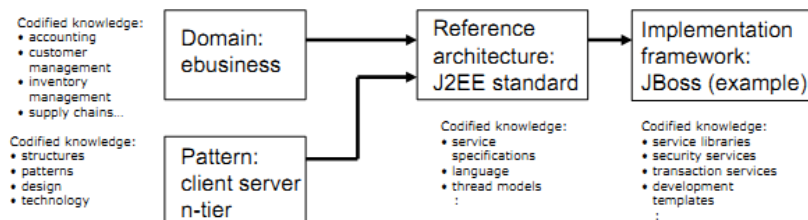
29



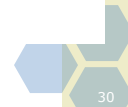
Domain Specific Patterns – 3



- Consider the emergence of the web oriented, business-to-business, IT application domain



- While the JBoss implementation framework is not the complete product architecture, it takes us a long way toward the solution



30



Tactics – 1



- ❖ **Often the type of problem or domain will suggest a particular pattern**
- ❖ **Once the architect has selected a pattern or ensemble of patterns that defines the overall structure of the system, many choices still remain**
- ❖ **Tactics provide finer grained structural details that allow more control over quality attribute responses**
- ❖ **Tactics do not have structure, but can be applied to various structures and may affect the structure of a system**



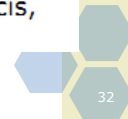
31



Tactics – 2



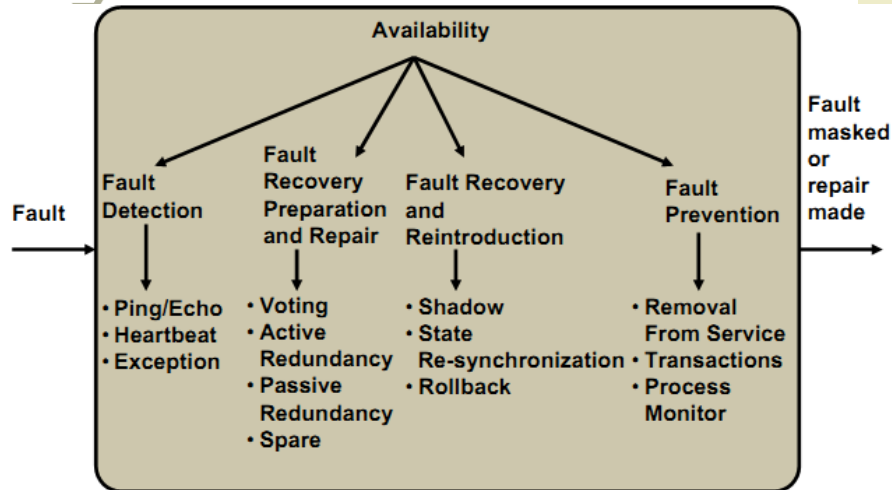
- Tactics are pre-packaged design options for the architect
 - For example, to promote availability, we might choose redundancy as a tactic
- Tactics can be used to refine other tactics
 - Example: redundancy can be refined to data and/or computational redundancy
- For a more complete treatment of tactics see:
 - Bass, L.; Clements, P. & Kazman, R. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley, 2003
 - Lattanze, A. Architecting Software Intensive Systems: A Practitioners Guide, New York, NY: Taylor and Francis, 2008



32



Summary of Availability Tactics

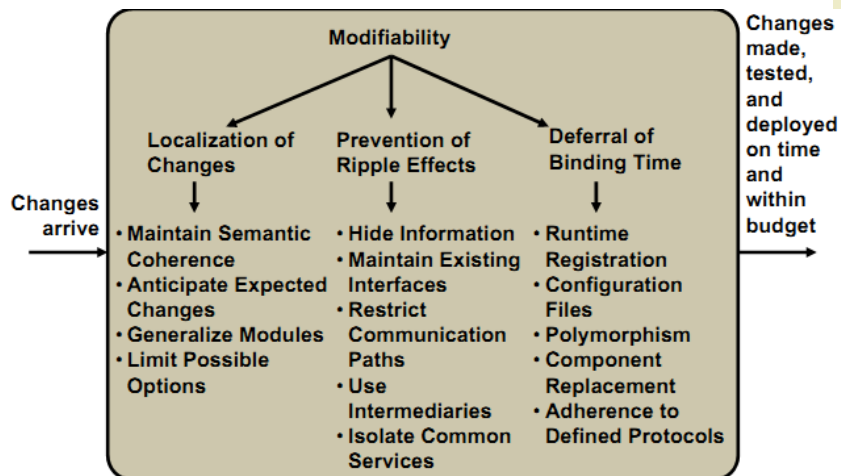


Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice*, Second Edition. Boston, MA: Addison-Wesley, 2003.

33



Summary of Modifiability Tactics

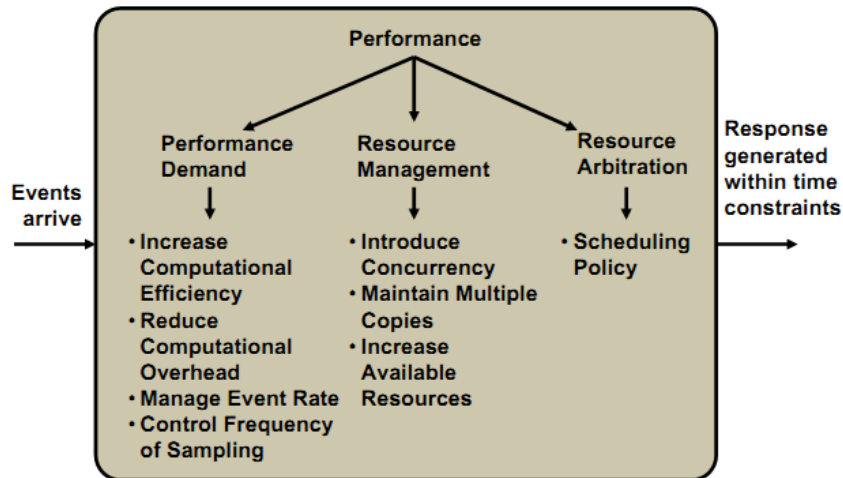


Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice*, Second Edition. Boston, MA: Addison-Wesley, 2003.

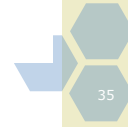
34



Summary of Performance Tactics



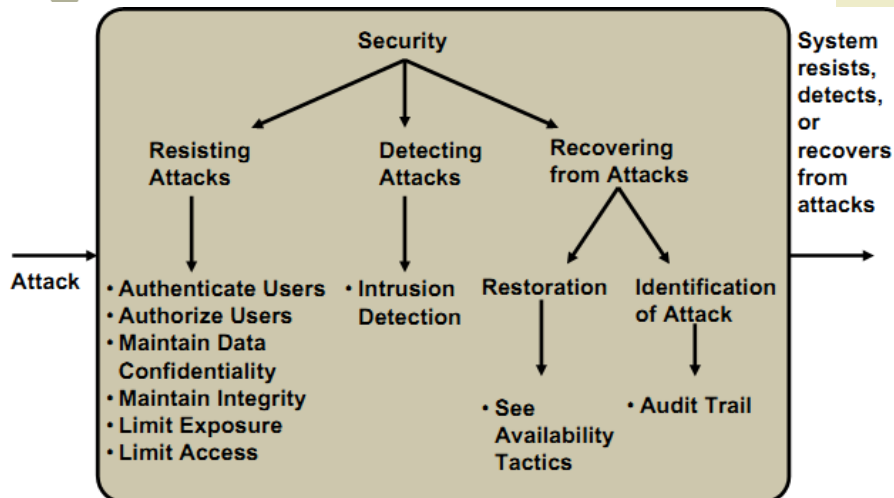
Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice*, Second Edition. Boston, MA: Addison-Wesley, 2003.



35



Summary of Security Tactics



Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice*, Second Edition. Boston, MA: Addison-Wesley, 2003.



36



Simple Tactics Illustration



Initial System Design Structure Client Server Pattern

- Promotes: Scalability
- Inhibits: Security, Performance, Availability

Applied Tactics

- Security: Authentication
- Performance: Multiple Servers
- Availability: Ping-Echo

This simple example should illustrate that tactics:

- do not have structure
- can be applied to various structures
- may affect the structure of a system



37



Summary



- In this session we
- introduced structures
 - structures are real, views are pictures of the structures
 - the kind of structures we can see and analyze depends on perspective
- discussed the essential qualities of structures
 - elements and relations
 - topology, perspective, semantics
 - typical usage, quality attributes promoted and inhibited



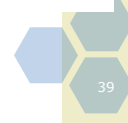
38



References



- Lattanze, A. *Architecting Software Intensive Systems: A Practitioners Guide*, New York, NY: Taylor and Francis, 2008
- Shaw, M.; Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*, Upper Saddle River, NJ: Prentice Hall, 1996
- Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice, Second Edition*. Boston, MA: Addison-Wesley, 2003
- Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern Oriented Software Architecture: A System of Patterns*. West Sussex England: John Wiley Ltd., 1996
- Gamma, E.; Helm, R.; Johnson R.; Vlissides, J. *Design Patterns*. Reading, MA: Addison-Wesley, 1994
- *Documenting Software Architecture: Views and Beyond*, Clements et al., Addison Wesley, 2003



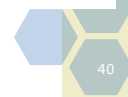
39



References



- Lattanze, A. *Architecting Software Intensive Systems: A Practitioners Guide*, New York, NY: Taylor and Francis, 2008
- Shaw, M.; Garlan, D. *Software Architecture: Perspectives on an Emerging Discipline*, Upper Saddle River, NJ: Prentice Hall, 1996
- Bass, L.; Clements, P. & Kazman, R. *Software Architecture in Practice, Second Edition*. Boston, MA: Addison-Wesley, 2003
- Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; Stal, M. *Pattern Oriented Software Architecture: A System of Patterns*. West Sussex England: John Wiley Ltd., 1996
- Gamma, E.; Helm, R.; Johnson R.; Vlissides, J. *Design Patterns*. Reading, MA: Addison-Wesley, 1994
- *Documenting Software Architecture: Views and Beyond*, Clements et al., Addison Wesley, 2003



40

